International Journal of Technology and Applied Science (IJTAS)

E-ISSN: 2230-9004 ● Website: www.ijtas.com ● Email: editor@ijtas.com

Bridging DevOps and ITSM: A ServiceNow-Based Framework for Automated Change Governance in CI/CD Pipelines

Praveen Chaitanya Jakku[1], Mohammed Shakeer Bandrevu[2]

[1,2]Independent Researcher

Abstract:
DevOps has changed how organizations build, test, and release software. Continuous integration and continuous delivery pipelines allow teams to move changes from source code to production with greater speed, consistency, and visibility. However, many enterprise organizations still manage production change through traditional IT Service Management processes such as manual change tickets, approval meetings, and disconnected release evidence. This creates a gap between modern CI/CD delivery and enterprise change governance.

This article proposes a ServiceNow-based framework for automated change governance in CI/CD pipelines. The framework connects DevOps pipeline metadata, test results, security scan outcomes, deployment evidence, risk classification, approval workflows, and post-deployment validation with ServiceNow change records. Instead of treating change management as a separate manual task, the proposed model embeds governance directly into the software delivery pipeline.

The article argues that DevOps and ITSM should not be viewed as opposing practices. DevOps supports automation, collaboration, and faster delivery, while ITSM provides accountability, traceability, and operational control. A ServiceNow-based governance layer can help organizations release software faster while maintaining audit readiness, compliance evidence, and production stability.

Keywords: DevOps, ITSM, ServiceNow, CI/CD, change management, automated governance, ITIL, release management, change approval.

1. Introduction

DevOps has become an important approach for organizations that want to deliver software faster without losing operational reliability. Modern software teams are expected to release changes frequently, respond quickly to defects, and support stable production systems. Continuous integration and continuous delivery pipelines help achieve this goal by automating build, test, security scan, packaging, and deployment activities.

However, enterprise software delivery is not only about speed. Production systems often support business operations, customer-facing services, sensitive data, financial transactions, and regulated workflows. A failed release can cause service outages, compliance issues, customer dissatisfaction, and operational disruption. For this reason, change management remains an important part of IT Service Management.

The challenge is that many traditional change management processes were created for slower release cycles. In a traditional model, teams manually create change requests, enter deployment details, attach evidence, wait for approvals, attend change meetings, and close the change after implementation. This process may work for monthly or quarterly releases, but it becomes difficult when teams use automated CI/CD pipelines and release more frequently.

This creates a practical gap between DevOps and ITSM. The CI/CD pipeline already knows important details such as the commit ID, build number, test results, artifact version, deployment status, approver,

and rollback outcome. However, the ServiceNow change record may still depend on manually entered information. This leads to duplicated work, delayed approvals, incomplete audit trails, and weak traceability.

DevOps research shows that DevOps is not only a toolset but a combination of practices, culture, automation, and collaboration between development and operations teams [1]. At the same time, CI/CD research shows that automated delivery brings clear benefits, but it also introduces challenges around testing, deployment control, and organizational adoption [2]. Therefore, the main question is not whether organizations should choose DevOps or ITSM. The real question is how ITSM can be modernized to support DevOps delivery.

This article proposes a ServiceNow-based framework for automated change governance in CI/CD pipelines. The framework uses the pipeline as the source of technical evidence and ServiceNow as the governance system of record. The objective is to reduce manual change effort while maintaining approval discipline, auditability, and production control.

2. Background: DevOps, ITSM, and Change Governance

DevOps aims to improve software delivery by connecting development, operations, testing, security, and business stakeholders. It encourages automation, shared ownership, faster feedback, and continuous improvement. Jabbari et al. describe DevOps as a set of definitions and practices that focus on collaboration between development and operations, supported by automation and process improvement [1].

Continuous integration and continuous delivery are key DevOps practices. They allow teams to validate every change through automated build and test stages before promoting it to higher environments. Shahin, Babar, and Zhu explain that CI/CD practices improve delivery speed and reliability, but they also require strong attention to tools, process maturity, testing quality, and deployment control [2].

ITSM focuses on managing IT services in a controlled and reliable manner. Change management is one of its most important practices because it helps organizations evaluate, approve, schedule, implement, and review production changes. In many enterprises, ServiceNow is used as the central platform for change management, incident management, problem management, configuration management, and service request workflows.

The relationship between DevOps and ITSM is sometimes misunderstood. DevOps is often seen as fast and flexible, while ITSM is seen as formal and control-focused. In reality, both are necessary. Galup, Dattero, and Quan argue that Agile, Lean, ITIL, and DevOps can work together when organizations use them as complementary approaches instead of treating them as competing models [3].

This is especially true for change management. DevOps does not remove the need for change governance. Instead, it changes how change governance should be performed. In a CI/CD environment, governance should be more automated, more evidence-driven, and more closely connected to the delivery pipeline.

3. Problem Statement

Traditional change management creates friction in CI/CD environments for several reasons.

First, manual change ticket creation duplicates work. Developers and DevOps engineers already provide details in Git commits, pull requests, build logs, test reports, security scan outputs, deployment manifests, and release notes. Re-entering the same information into ServiceNow wastes time and increases the chance of inaccurate or incomplete records.

Second, approval delays can slow down delivery. If every production change follows the same manual approval path, even low-risk changes may wait unnecessarily. This may cause teams to bundle many small changes into larger releases. Larger releases are usually harder to test, harder to review, and harder to roll back.

Third, audit evidence is often scattered across multiple systems. Build results may be in Jenkins, source code in Git, test results in a testing platform, security results in a scanning tool, deployment logs in the

pipeline, and approval records in ServiceNow. During audits or incidents, teams may spend significant time reconstructing what happened.

Fourth, risk classification can be inconsistent. One engineer may classify a change as low risk, while another engineer may classify a similar change as medium risk. Without clear rules and automated evidence, risk decisions depend too much on individual judgment.

Fifth, the approved change record may not always match the actual deployment. A change request may describe one build version, but the pipeline may deploy a later artifact, a different container image, or a modified configuration. This weakens traceability and creates problems during incident investigation.

Continuous delivery literature also shows that while frequent releases can provide major benefits, adoption is not only a technical matter. Organizations must handle process, cultural, architectural, and governance challenges [4], [5]. Therefore, disconnected change management becomes a barrier to mature DevOps adoption.

The problem is not change management itself. The real problem is manual and disconnected change management. Governance must move closer to the pipeline.

4. Proposed ServiceNow-Based Framework

The proposed framework uses ServiceNow as the governance layer and the CI/CD pipeline as the execution and evidence layer. The framework is designed around five main stages: change initiation, pipeline validation, automated change record creation, risk-based approval, and deployment validation.

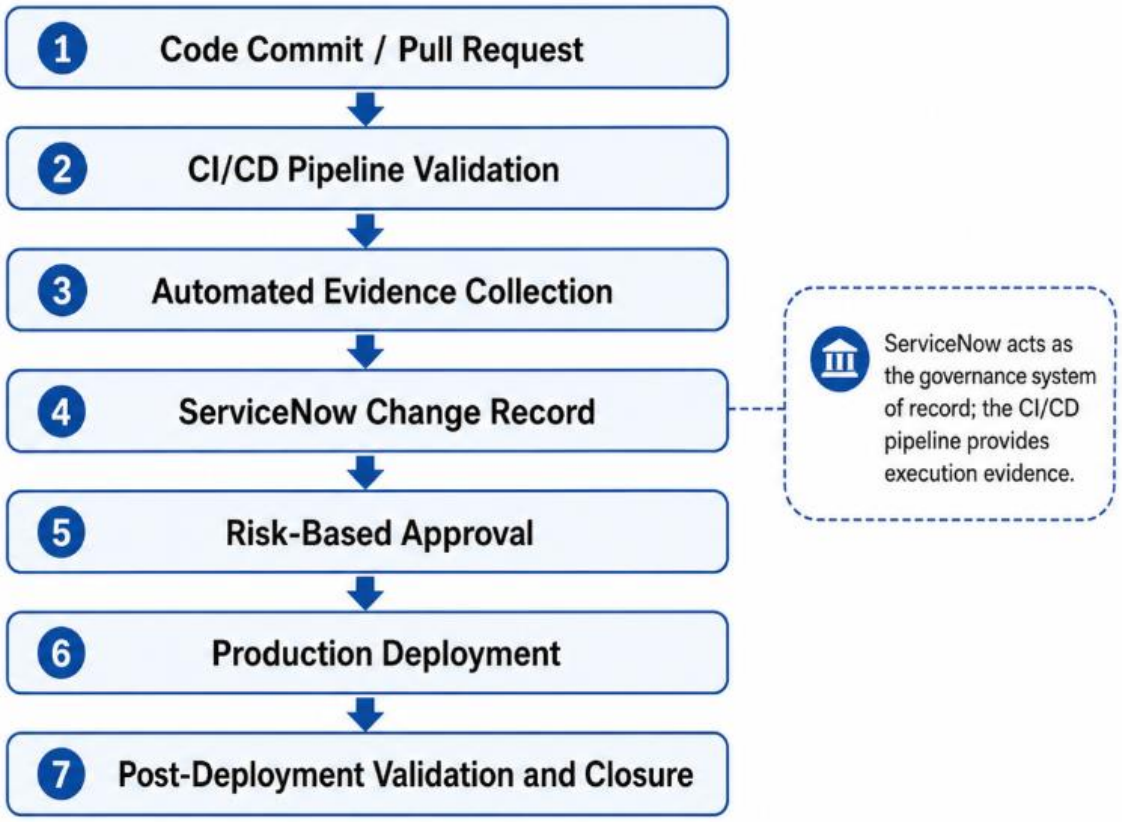

Figure 1. Proposed ServiceNow-based automated change governance workflow for CI/CD pipelines.

Figure 1 shows how change governance can be embedded directly into the CI/CD lifecycle. The process begins with a code commit or pull request, followed by pipeline validation and automated evidence collection. The collected evidence is then used to create or update a ServiceNow change record, where

risk-based approval is applied before production deployment. After deployment, validation results are sent back to ServiceNow to support closure, rollback, or incident linkage.

4.1 Change Initiation

The change begins when a developer commits code or raises a pull request linked to a story, defect, or task. At this stage, the pipeline captures source control metadata such as repository name, branch, commit ID, pull request number, author, reviewer, and work item reference.

This information helps answer basic governance questions: What changed? Why was it changed? Who reviewed it? Which business requirement or defect is connected to the change?

4.2 Pipeline Validation

After the code is merged or promoted, the CI/CD pipeline validates the change. This may include unit testing, integration testing, static code analysis, dependency scanning, container image scanning, infrastructure-as-code validation, configuration checks, and deployment readiness checks.

These validations provide objective evidence about release quality. DevSecOps research highlights that security adoption in DevOps requires both technical controls and process integration, not just separate security reviews at the end of delivery [6]. Therefore, security and quality evidence should become part of the automated change record.

In cloud-native environments such as Amazon EKS, pipeline validation should also consider least-privilege access, workload isolation, secrets management, admission control, and continuous visibility as part of production readiness and secure delivery governance [8].

4.3 Automated Change Record Creation

Before production deployment, the pipeline creates or updates a ServiceNow change record through API-based integration. The change record should not be a manually written summary. It should be populated with structured pipeline evidence.

A useful change record may include:

- Application or service name
- Environment
- Change type
- Business service or configuration item
- Pipeline name and run ID
- Git commit ID
- Pull request or merge request reference
- Build number
- Artifact version or container image tag
- Test result summary
- Security scan result summary
- Deployment plan
- Rollback plan
- Requested deployment window
- Risk score
- Approver details
- Deployment outcome
- Post-deployment validation status

This makes ServiceNow the governance system of record, while the pipeline remains the execution system of truth.

4.4 Risk-Based Approval

The framework should not treat every change the same way. A small configuration update to a low-risk service should not require the same review as a database migration, identity policy change, or production network modification.

ServiceNow can apply risk-based approval rules using information from the pipeline and the configuration management database. Risk can be evaluated using factors such as service criticality, production impact, number of services affected, database changes, infrastructure changes, test results, security findings, recent incidents, and rollback readiness.

A low-risk standard change may be automatically approved if all conditions are met. A medium-risk change may require service owner approval. A high-risk change may require CAB review, security review, or additional validation. Emergency changes may follow a faster approval path but should still include post-implementation review.

This model supports both DevOps speed and ITSM control.

4.5 Deployment Validation and Closure

After approval, the pipeline deploys the change. Once deployment is complete, the pipeline runs smoke tests and checks monitoring signals. The result is sent back to ServiceNow.

If the deployment succeeds and the service remains healthy, the change can be closed after validation. If the deployment fails, the pipeline should update the change record, trigger rollback, and create or link an incident if needed. This creates a complete operational trail from code commit to production outcome.

5. Implementation Model

A practical implementation should begin with one application and one pipeline. Trying to automate all change processes at once can create confusion and resistance. A phased approach is more realistic.

First, the organization should identify an application that already has a stable CI/CD pipeline, clear service ownership, reliable automated tests, and a known rollback process. This type of application is a good candidate for automated change governance.

Second, the team should define the minimum evidence required for a production change. This may include commit ID, build result, artifact version, test summary, security scan summary, implementation plan, rollback plan, deployment window, and service owner approval.

Third, pipeline stages should be mapped to ServiceNow fields. For example, the Jenkins build number can map to the pipeline run ID, the Git commit can map to the source reference, the container image tag can map to the artifact version, and the smoke test result can map to post-deployment validation.

Fourth, change models should be clearly defined. A standard change may require all tests to pass, no critical vulnerabilities, no database changes, no active incident on the service, and a tested rollback process. A high-risk change may include production database changes, network changes, identity and access management changes, or deployment to a critical service.

Fifth, the pipeline should integrate with ServiceNow using secure API credentials or approved integration connectors. The integration user should follow least-privilege access. It should be allowed to create, update, and read only the necessary change records.

Sixth, the pipeline should wait for the ServiceNow change state before production deployment. If the change is automatically approved, the pipeline proceeds. If manual approval is required, the pipeline pauses. If the change is rejected, the deployment stops.

Seventh, the pipeline should update the change after deployment. The change should not be closed simply because deployment started. It should be closed after deployment validation is complete.

This model supports gradual adoption and reduces the risk of disrupting existing release processes.

6. Expected Benefits

The proposed framework provides several benefits.

The first benefit is reduced manual work. Engineers no longer need to copy pipeline information into change records. This saves time and improves record accuracy.

The second benefit is faster release flow. Low-risk changes can move through standard or automated approval paths, while high-risk changes still receive proper review. This supports DevOps delivery speed without removing governance.

The third benefit is improved audit readiness. Change records contain technical evidence such as commit ID, build number, test result, artifact version, approval status, deployment result, and rollback evidence. This makes audit preparation easier and more reliable.

The fourth benefit is better incident response. If an incident occurs after deployment, teams can quickly identify the related change, artifact version, deployment time, approval history, and validation result. This can reduce investigation time and support faster recovery.

The fifth benefit is stronger collaboration between DevOps and ITSM teams. ServiceNow becomes part of the delivery lifecycle rather than a separate ticketing system. DevOps teams gain a clearer approval path, and ITSM teams gain better visibility into release activity.

The sixth benefit is consistent risk classification. Automated rules reduce subjective decision-making and help ensure similar changes are handled in a similar way.

Leite et al. note that DevOps adoption involves both concepts and challenges, including automation, collaboration, monitoring, measurement, and organizational change [7]. The proposed framework supports these areas by connecting automation with governance and operational feedback.

7. Challenges and Limitations

The framework also has limitations.

Data quality is a major challenge. If service ownership, configuration items, application names, and environment mappings are inaccurate, automated change records will also be inaccurate. A weak CMDB can reduce the value of automated governance.

Rule design is another challenge. If risk rules are too strict, teams may experience the same delays as manual change management. If rules are too relaxed, risky changes may be approved too easily. The organization must review risk rules regularly and adjust them based on production outcomes.

Toolchain complexity can also make implementation difficult. Large organizations may use multiple source control systems, CI/CD platforms, artifact repositories, cloud platforms, and monitoring tools. Integration should be standardized but introduced gradually.

Security must be handled carefully. Pipeline credentials used to access ServiceNow should follow least privilege. API tokens and secrets should be stored securely, rotated periodically, and monitored for misuse.

Organizational adoption may take time. Change managers may worry that automation removes control, while engineers may worry that ServiceNow integration will slow delivery. These concerns can be reduced through pilot implementation, transparent reporting, and clear separation between low-risk and high-risk change paths.

Finally, automation should not be used to hide poor process design. Automating an inefficient process may only make the inefficiency happen faster. Before integration, organizations should simplify unnecessary approval steps and clearly define which changes require human review.

8. Example Scenario

Consider a team deploying a microservice to a production Kubernetes environment. A developer merges a pull request linked to a user story. The CI/CD pipeline starts automatically, builds the application, runs automated tests, scans dependencies, creates a container image, and prepares the deployment.

Before production deployment, the pipeline creates a ServiceNow change record. The record includes the commit ID, build number, image tag, test result, scan result, deployment plan, rollback command, and service owner. ServiceNow checks the change model.

Since the change does not include a database migration, all tests passed, no critical vulnerabilities were found, and rollback is available, the change is classified as low to medium risk. The service owner approves the change in ServiceNow. The pipeline continues deployment.

For Kubernetes-based microservices, deployment governance should also account for rollout strategy, readiness checks, observability signals, rollback preparedness, and recovery validation, because a technically completed deployment does not always mean the service is healthy for users [9].

After deployment, smoke tests pass and monitoring shows normal service behavior. The pipeline updates the ServiceNow record and closes the change after validation.

If the deployment fails, the pipeline triggers rollback, updates the change record, and creates or links an incident. This gives the organization a complete operational trail from code commit to production release.

9. Conclusion

DevOps and ITSM should not be treated as opposing approaches. DevOps improves delivery speed through automation, feedback, and collaboration. ITSM provides structure, accountability, service ownership, and operational control. The real challenge is connecting these practices in a way that supports both speed and stability.

This article proposed a ServiceNow-based framework for automated change governance in CI/CD pipelines. The framework uses the pipeline as the source of technical evidence and ServiceNow as the governance system of record. By connecting build results, test evidence, security checks, artifact details, approval rules, deployment outcomes, and post-deployment validation, organizations can reduce manual effort and improve release control.

The proposed model supports risk-based change approval. Low-risk changes can move faster through standard or automated workflows, while high-risk changes continue to receive manual review. This allows enterprises to modernize change management without abandoning governance.

For organizations using ServiceNow and CI/CD pipelines, this framework provides a practical path toward evidence-driven change management. It helps teams release software faster, maintain audit readiness, improve incident response, and protect production stability.

REFERENCES:

[1] R. Jabbari, N. B. Ali, K. Petersen, and B. Tanveer, “What is DevOps? A systematic mapping study on definitions and practices,” Proceedings of the Scientific Workshop Proceedings of XP2016, 2016, pp. 1–11. DOI: 10.1145/2962695.2962707.

[2] M. Shahin, M. A. Babar, and L. Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” IEEE Access, vol. 5, pp. 3909–3943, 2017. DOI: 10.1109/ACCESS.2017.2685629.

[3] S. Galup, R. Dattero, and J. Quan, “What do agile, lean, and ITIL mean to DevOps?” Communications of the ACM, vol. 63, no. 10, pp. 48–53, 2020. DOI: 10.1145/3372114.

[4] L. Chen, “Continuous Delivery: Huge Benefits, but Challenges Too,” IEEE Software, vol. 32, no. 2, pp. 50–54, 2015. DOI: 10.1109/MS.2015.27.

[5] L. Chen, “Continuous Delivery: Overcoming Adoption Challenges,” Journal of Systems and Software, vol. 128, pp. 72–86, 2017. DOI: 10.1016/j.jss.2017.02.013.

[6] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, “Challenges and solutions when adopting DevSecOps: A systematic review,” Information and Software Technology, vol. 141, 2022. DOI: 10.1016/j.infsof.2021.106700.

[7] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, “A Survey of DevOps Concepts and Challenges,” ACM Computing Surveys, vol. 52, no. 6, pp. 1–35, 2019. DOI: 10.1145/3359981.

[8] P. C. Jakku, "A Zero Trust Reference Architecture for Production-Ready Amazon EKS Environments," International Journal of Innovative Research and Creative Technology, vol. 8, no. 2, pp. 1–9, 2022. DOI: https://doi.org/10.62970/IJIRCT.v8.i2.2605004
[9] P. C. Jakku, "Resilient Kubernetes Deployment Strategies for Microservices: A Practical Reliability Model," International Journal for Multidisciplinary Research, vol. 3, no. 6, 2021. DOI: https://doi.org/10.36948/ijfmr.2021.v03i06.77664